HIGHER-ORDER THEORIES

ANDRÉ HIRSCHOWITZ AND MARCO MAGGESI

ABSTRACT. We extend our approach to abstract syntax (with binding constructions) through modules and linearity. First we give a new general definition of arity, yielding the companion notion of signature. Then we obtain a modularity result as requested by [GU03]: in our setting, merging two extensions of syntax corresponds to building an amalgamated sum. Finally we define a natural notion of equation concerning a signature and prove the existence of an initial semantics for a so-called *representable* signature equipped with a set of equations.

1. INTRODUCTION

1.1. **Modules for modular higher-order syntax and semantics.** Many programming or logical languages allow constructions which bind variables and this higher-order feature causes much trouble in the formulation, the understanding and the formalization of the theory of these languages. For instance, there is no universally accepted discipline for such formalizations: that is precisely why the POPLmark Challenge [ABF+05] offers benchmarks for testing old and new approaches. Although this problem may ultimately concern typed languages and their operational semantics, it already concerns untyped languages equipped with an equational semantics. Indeed, even at the informal level, there is not yet a universally adopted notion of higher-order theory.

The goal of this work is to extend, in particular towards equational semantics, our approach to higher-order abstract syntax [HM07], based on modules and linearity. First of all, we give a new general definition of arity, yielding the companion notion of signature. The notion is coined in such a way to induce a companion notion of representation of an arity (or of a signature) in a monad: such a representation is a morphism among modules over the given monad, so that an arity simply assigns two modules to each monad. Then we explain how our approach enjoys modularity in the sense introduced by [GU03]: in our view, a syntax is or yields a monad equipped with some module morphisms; such enriched monads form a category in which merging two extensions of syntax corresponds to building an amalgamated sum. Finally, we define a general notion of equation for a given signature (or syntax) Σ and we prove the corresponding theorem for initial semantics. As for arities, our notion of equation is coined in such a way to induce the notion of whether the equation is satisfied by an arbitrary representation of Σ in a monad R: this will be when two morphisms of modules over R coincide. Accordingly an equation assigns to each representation of Σ a pair of morphisms of modules (with common domain and codomain).

1.2. **Examples.** Such a proposal should definitely offer a convincing picture concerning the paradigmatic example of the lambda-calculus. In our previous work [HM07] we treated the examples of the untyped lambda-calculus modulo α-equivalence and $\alpha\beta\eta$-equivalence. In section 6 we revisit these two examples. We then give various other examples obtained by adding features to the lambda-calculus: differentiation, parallelism, explicit substitution.

1.3. **Future work.** In the present work we give a notion of untyped higher-order theory, and we observe the existence of an initial representation for *algebraic* signatures. This has to be extended to more general signatures, including the signature of the lambda-calculus with explicit substitution, as pioneered in [GU03], in order to reach a satisfactory state for the untyped equational setting. The point of view proposed here will also be accommodated to deal with languages with a fixed set of types (for an example see [Zsi06]), or to model operational semantics through monads on the category of preordered sets, or both. It should also be extended to the case of dependent types.

1.4. **Organization of the paper.** Section 2 gives a succinct account about modules over a monad. Our new definitions of (higher-order) arity and signature are given in section 3. We propose a solution to the problem of modularity in section 4. Section 5 develops the theory of equations for our notion of syntax and contains our theorem about initial semantics. Section 6 treats some explicit examples. The last section discusses related works.

2. Modules over Monads

We recall only the definition and some basic facts about (right) modules over a monad. See [HM07] for a more extensive introduction on this topic.

Let C be a category. A monad over C is a monoid in the category $\mathsf{C} \to \mathsf{C}$ of endofunctors of C, i.e., a triple $R = \langle R, \mu, \eta \rangle$ given by a functor $R\colon \mathsf{C} \to \mathsf{C}$, and two natural transformations $\mu\colon R^2 \to R$ and $\eta\colon I \to R$ such that the following diagrams commute:

$$
\begin{array}{ccc}
R^3 \xrightarrow{\;R\mu\;} R^2 & \qquad & I \cdot R \xrightarrow{\;\eta R\;} R^2 \xleftarrow{\;R\eta\;} R \cdot I \\
\mu R \downarrow \qquad \downarrow \mu & & {}_{1_R}\searrow \quad \downarrow \mu \quad \swarrow {}_{1_R} \\
R^2 \xrightarrow{\;\mu\;} R & & R
\end{array}
$$

Let R be a monad over C.

Definition 2.1 (Right modules)**.** A right R-module is given by a functor $M\colon \mathsf{C} \to \mathsf{D}$ equipped with a natural transformation $\rho\colon M \cdot R \to M$, called *action*, which is compatible with the monad composition and identity:

$$
\begin{array}{ccc}
M \cdot R^2 \xrightarrow{\;M\mu\;} M \cdot R & \qquad & M \cdot R \xleftarrow{\;M\eta\;} M \cdot I \\
\rho R \downarrow \qquad \downarrow \rho & & \rho \downarrow \quad \swarrow {}_{1_M} \\
M \cdot R \xrightarrow{\;\rho\;} M & & M
\end{array}
$$

We will refer to the category D as the *range* of M.

We say that a natural transformation of right R-modules $\tau\colon M \to N$ is *linear* if it is compatible with action:

$$
\begin{array}{ccc}
M \cdot R & \xrightarrow{\ \tau R\ } & N \cdot R \\
{\scriptstyle \rho_M}\downarrow & & \downarrow{\scriptstyle \rho_N} \\
M & \xrightarrow{\ \tau\ } & N
\end{array}
$$

We take linear natural transformations as morphisms among right modules having the same range D. It can be easily verified that we obtain in this way a category that we denote $\mathrm{Mod}^{\mathsf{D}}(R)$.

There is an obvious corresponding definition of left R-modules that we do not need to consider in this paper. From now on, we will write R-modules instead of right R-modules for brevity.

Example 2.2. Let us show some trivial examples of modules:
  (1) Every monad R is a module over itself, which we call the *tautological* module.
  (2) For any functor $F\colon \mathsf{D} \to \mathsf{E}$ and any R-module $M\colon \mathsf{C} \to \mathsf{D}$, the composition $F \cdot M$ is a R-module (in the evident way).
  (3) For every object $W \in \mathsf{D}$ we denote by $\underline{W}\colon \mathsf{C} \to \mathsf{D}$ the constant functor $\underline{W} := X \mapsto W$. Then $\underline{W}$ is trivially a R-module since $\underline{W} = \underline{W} \cdot R$.

Limits and colimits in the category of right modules can be constructed point-wise. For instance:

Lemma 2.3 (Limits and colimits of modules)**.** *If D is complete (resp. cocomplete), then $\mathrm{Mod}^{\mathsf{D}}(R)$ is complete (resp. cocomplete).*

In particular, we will often make use of the fact that, if the range category D is cartesian, then the category $\mathrm{Mod}^{\mathsf{D}}(R)$ is also cartesian.

For our purposes, one important example of module is given by the following general construction. Let C be a category with finite colimits and a final object $*$.

Definition 2.4 (Derivation)**.** For any R-module M with range D, the *derivative* of M is the functor $M := X \mapsto M(X + *)$. Derivation can be iterated, we denote by $M^{(k)}$ the k-th derivative of M.

Proposition 2.5. *Derivation yields an endofunctor of $\mathrm{Mod}^{\mathsf{D}}(R)$. Moreover, if D is a cartesian category, derivation is a cartesian endofunctor of $\mathrm{Mod}^{\mathsf{D}}(R)$.*

In the case $\mathsf{C} = \mathsf{D} = \mathsf{Set}$, the functor M' is given by $M' := X \mapsto M(X+*)$, where $X+*$ denotes the set obtained by adding a new point to X. Moreover, we have a natural *evaluation morphism*

$$\mathsf{eval}\colon M' \times R \longrightarrow M.$$

which is R-linear. This allows us to interpret the derivative M' as the "module M with one formal parameter added". Higher-order derivatives have analogous morphisms (that we still denote with eval)

$$\mathsf{eval}\colon M^{(b)} \times R^b \longrightarrow M.$$

where $\mathsf{eval}(t, m_1, \ldots, m_b) \in M(X)$ is obtained by substituting $m_1, \ldots, m_b \in R(X)$ in the successive stars of $t \in M^{(b)}(X) = M(X + * + \cdots + *)$.

We already introduced the category $\mathrm{Mod}^{\mathsf{D}}(R)$ of modules with fixed base R and range D. It it often useful to consider a larger category which collects modules with different bases. To this end, we need first to introduce the notion of pull-back.

Definition 2.6 (Pull-back)**.** Let $f \colon R \to S$ be a morphism of monads and M a S-module. The action

$$M \cdot R \xrightarrow{Mf} M \cdot S \xrightarrow{\rho} M$$

defines a R-module which is called *pull-back* of M along f and noted f^*M. It can be easily verified that a S-linear natural transformation $g \colon M \to N$ is also a R-linear natural transformation $f^*g \colon f^*M \to f^*N$ and that $f^* \colon \mathrm{Mod}^{\mathsf{D}}(S) \to \mathrm{Mod}^{\mathsf{D}}(R)$ is a functor.

It can be easily verified that pull-back is well-behaved with respect to many important constructions. In particular:

Proposition 2.7. *Pull-back commutes with products and with derivation.*

Definition 2.8. Given a list of non negative integers $(a) = (a_1, \ldots, a_n)$ we denote by $M^{(a)} = M^{(a_1, \ldots, a_n)}$ the module $M^{(a_1)} \times \cdots \times M^{(a_n)}$. Observe that, when $(a) = ()$ is the empty list, we have $M^{()} = *$ the final module.

Definition 2.9 (The large module category)**.** We define the large module category LMod as follows:

- its objects are pairs (R, M) of a monad R and a R-module M.
- a morphism from (R, M) to (S, N) is a pair (f, m) where $f \colon R \longrightarrow S$ is a morphism of monads, and $m \colon M \longrightarrow f^*N$ is a morphism of R-modules. The category LMod comes equipped with a forgetful functor to the category of monads, given by the projection $(R, M) \mapsto R$.

3. HALF-ARITIES AND SIGNATURES

In this section, we improve our approach to higher-order syntax [HM07], and give a new notion of arity. The destiny of an arity is to have representations in monads. A representation of an arity a in a monad R will be a morphism between two modules $\mathrm{dom}(a, R)$ and $\mathrm{codom}(a, R)$. For instance, in the case of the arity a of app_2, we have $\mathrm{dom}(a, R) := R^2$ and $\mathrm{codom}(a, R) := R$. Hence an arity consists of two halves, each of which assigns to each monad R a module over R in a functorial way.

Definition 3.1 (Arity)**.** A *half-arity* is a right-inverse functor to the projection from the category LMod to the category Mon of monads. An *arity* is a pair of two half-arities. The arity (a, b) is denoted $a \to b$; a and b are called respectively the domain and the codomain of $a \to b$.

Example 3.2.   (1) The assignment $R \mapsto R$ is a half-arity which we denote by Θ.
  (2) The assignment $R \mapsto *$, where $*$ denotes the final module over R is a half-arity which we denote by $*$.

(3) Given a half-arity a, for each non-negative integer n, the assignment $R \mapsto a(R)^{(n)}$ is a half-arity which we denote by $a^{(n)}$. As usual, we also set $a' := a^{(1)}$ and $a'' := a^{(2)}$.

(4) Given two half-arities a and b, the assignment $R \mapsto a(R) \times b(R)$ is a half-arity which we denote by $a \times b$.

(5) For each non-negative integer n, the assignment $R \mapsto R^n$ is a half-arity which we denote by Θ^n.

(6) For each sequence of non-negative integers s, the assignment $R \mapsto R^{(s)}$ is a half-arity which we denote by $\Theta^{(s)}$.

(7) The assignment $R \mapsto R \cdot R$ is a half-arity which we denote by $\Theta \cdot \Theta$. Of course we have plenty of such composite half-arities.

We denote by Ar the set of arities.

Definition 3.3 (Algebraic and raw arities)**.** Half-arities of the form $\Theta^{(s)}$ and arities of the form $\Theta^{(s)} \to \Theta^{(t)}$ are said algebraic. An arity of the form $H \to \Theta$ is said raw.

These algebraic arities are slightly more general than those in [FPT99], which are precisely algebraic raw arities. In particular we have an algebraic arity $(\Theta \to \Theta')$ for the app_1 construction given in section 6.2.

Definition 3.4 (Signatures)**.** We define a signature $\Sigma = (O, \alpha)$ to be a family of arities $\alpha \colon O \to \mathrm{Ar}$. A signature is said to be raw (resp. algebraic) if it consists of raw (resp. algebraic) arities.

Definition 3.5 (Representation of an arity, of a signature)**.** Given a monad R over Set, we define a representation of the arity a in R to be a module morphism from $\mathrm{dom}(a, R)$ to $\mathrm{codom}(a, R)$; a representation of a signature S in R consists of a representation in R for each arity in S.

Example 3.6. The usual $\mathsf{app} \colon \mathsf{LC}^2 \to \mathsf{LC}$ (see section 6.3) is a representation of $\Theta^2 \to \Theta$ into LC. A representation of $* \to \Theta''$ in LC is given by the app_0 construction $(\mathsf{app}_0(X) := \mathsf{app}(\mathsf{var}(*_{X+*}), \mathsf{var}(*_X)))$.

Definition 3.7 (The category of representations)**.** Given a signature $\Sigma = (O, \alpha)$, we build the category Mon^Σ of representations of Σ as follows. Its objects are monads equipped with representations of Σ. A morphism from (M, r) to (N, s) is a morphism from M to N compatible with the representations in the sense that, for each o in O, the following diagram commutes:

$$
(1) \qquad
\begin{array}{ccc}
\mathrm{dom}(\alpha(o), M) & \longrightarrow & \mathrm{codom}(\alpha(o), M) \\
\downarrow & & \downarrow \\
f^* \, \mathrm{dom}(\alpha(o), N) & \longrightarrow & f^* \, \mathrm{codom}(\alpha(o), N)
\end{array}
$$

where the horizontal arrows come from the representations and the vertical arrows come from the functoriality of half-arities.

Proposition 3.8. *These morphisms, together with the obvious composition, turn* Mon^Σ *into a category which comes equipped with a forgetful functor to the category of monads.*

Definition 3.9. A signature Σ is said representable if the category Mon^Σ has an initial object.

Theorem 3.10. *Algebraic signatures are representable.*

For more details we refer to our paper [HM07] (theorems 1 and 2).

Remark 3.11. There is a slightly more general notion of arity and signature. Roughly speaking, a Σ-arity will be a pair of Σ-modules (see below section 5). Such a Σ-arity may be added to Σ, yielding a bigger signature. This picture allows to consider partially defined constructions, like the predecessor. We leave this extension for future work.

4. Modularity

It has been stressed [GU03] that the standard approach (via algebras) to higher-order syntax lacks modularity. In the present section we show in which sense our approach via modules enjoys modularity.

Suppose that we have a signature $\Sigma = (O, a)$ and two subsignatures Σ' and Σ'' covering Σ in the obvious sense, and let Σ_0 be the intersection of Σ' and Σ''. Suppose that these four signatures are representable (for instance because Σ is algebraic). Modularity would mean that the corresponding diagram of monads

$$
\begin{array}{ccc}
\hat{\Sigma}_0 & \longrightarrow & \hat{\Sigma}' \\
\downarrow & & \downarrow \\
\hat{\Sigma}'' & \longrightarrow & \hat{\Sigma}
\end{array}
$$

is cocartesian. The observation of [GU03] is that the diagram of raw monads is, in general, not cocartesian. Since we do not want to change the monads, in order to claim for modularity, we will have to consider a category of enriched monads. Here by enriched monad, we mean a monad equipped with some additional structure.

Our solution to this problem goes through the following category WRep:

- An object of WRep is a triple (R, Σ, r) where R is a monad, Σ a signature, and r is a representation of Σ in R.
- A morphism in WRep from $(R, (O, a), r)$ to $(R', (O', a'), r')$ consists of a map $i := O \to O'$ compatible with a and a' and a morphism m from (R, r) to $(R', i^*(r'))$, where, for i injective, $i^*(r')$ should be understood as the restriction of the representation r' to the subsignature (O, a).
- It is easily checked that the obvious composition turns WRep into a category.

Now for each signature Σ, we have an obvious functor from Mon^Σ to WRep, through which we may see $\hat{\Sigma}$ as an object in WRep. Furthermore, an injection $i \colon \Sigma_1 \to \Sigma_2$ obviously yields a morphism $i_* := \hat{\Sigma}_1 \to \hat{\Sigma}_2$ in WRep. Hence our 'cocartesian' square of signatures as described above yields a square in WRep. The proof of the following statement is straightforward.
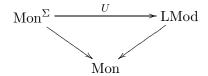
Proposition 4.1. *Modularity holds in* WRep*, in the sense that given a 'cocartesian' square of signatures as described above, the associated square in* WRep *is cocartesian again.*

There is a stronger statement in a category Rep with the same objects and more morphisms. Roughly speaking, in Rep, a morphism from (R, Σ, r) to (R', Σ', r') is a compatible pair of a monad morphism and a "vertical" functor from $Mon^{\Sigma'}$ to Mon^{Σ}. We plan to describe this more carefully in some future work.

5. The Category of Half-Equations

In this section, we are given a signature Σ, and we build the category where our equations will live.

Definition 5.1 (The category of half-equations)**.** We define a Σ-module U to be a functor from the category of representations of Σ to the category LMod commuting with the forgetful functors to the category Mon of monads.

$$\text{Mon}^{\Sigma} \xrightarrow{\quad U \quad} \text{LMod}$$
$$\searrow \qquad \swarrow$$
$$\text{Mon}$$

We define a morphism of Σ-modules to be a natural transformation which becomes the identity when composed with the forgetful functor. We call these morphisms "half-equations".

These definitions yield a category which we call the category of Σ-modules or half-equations.

Example 5.2. To each half-arity a is associated, by composition with the projection from Mon^{Σ} to Mon, a Σ-module still denoted a. In particular we have a Σ-module Θ. Accordingly, to each construction $c \in \Sigma$ with arity $a \rightarrow b$ is associated in the obvious way a morphism of Σ-modules, still denoted c, from a to b.

Example 5.3. Since derivation is a "vertical" endofunctor in LMod, it acts on Σ-modules. In particular we also have a family of Σ-modules $\Theta^{(n)}$.

Proposition 5.4. *The category of half-equations is cartesian.*

Example 5.5. To each application $f \colon [1, \ldots, p] \longrightarrow [1, \ldots, q]$ is associated a half-equation: $f_* \colon T^{(p)} \longrightarrow T^{(q)}$ (which we call renaming along f).

Definition 5.6 (Equations)**.** We define an equation for Σ to be a pair of half-equations with common source and target. We also write $e_1 = e_2$ for the equation (e_1, e_2).

Example 5.7. In case Σ consists of the two constructions abs and app_1 (cfr. section 6.2) with respective arities $\Theta' \rightarrow \Theta$ and $\Theta \rightarrow \Theta'$, the β equation is $\mathsf{app}_1 \cdot \mathsf{abs} = \mathrm{Id}_{\Theta'}$, while the η equation is $\mathsf{abs} \cdot \mathsf{app}_1 = \mathrm{Id}_{\Theta}$.

Definition 5.8 (Satisfying equations)**.** We say that a representation r of Σ in a monad M satisfies the equation $e_1 = e_2$ if $e_1(r) = e_2(r)$. If E is a set of equations for Σ, we say that a representation r of Σ in a monad M satisfies E (or is a representation of (Σ, E)) if it satisfies each equation in E. We define the category of representations of (Σ, E) to be the full subcategory in the category of representations of Σ whose objects are representations of (Σ, E).

Theorem 5.9 (Initial representation of (Σ, E))**.** *Given a set E of equations for a representable signature Σ, the category of representations of (Σ, E) has an initial object.*

Sketch. We build the monad S for our initial representation as a quotient of $\hat{\Sigma}$ (see theorem 3.10). For each set X, we define an equivalence relation r_X on $\hat{\Sigma}(X)$ as follows: $r_X(a, b)$ means that for any representation ρ of (Σ, E) in a monad M, $i_X(a)$ equals $i_X(b)$, where $i \colon \hat{\Sigma} \longrightarrow M$ is the initial functor associated to ρ. We check easily that this is an equivalence relation, that the corresponding collection of quotients inherits the structure of monad, and that this quotient monad has the required universal property. □

6. Examples

We now illustrate the previous notions through some well-known examples.

6.1. **Monoids.** We first consider an example of first-order syntax with equations. Given a set X, let us denote by $M(X)$ the free monoid built over X. This is a classical example of monad over the category of (small) sets. The monoid structure gives us, for each set X, two maps $m_X \colon M(X) \times M(X) \to M(X)$ and $e_X \colon * \to M(X)$ given by the product and the identity respectively. It can be easily verified that $m \colon M^2 \longrightarrow M$ and $e \colon * \to M$ are M-linear natural transformations. In other words $(M, \rho) = (M, \{m, e\})$ constitutes a representation of the signature $\Sigma = \{m \colon \Theta^2 \to \Theta, e \colon * \to \Theta\}$.

In the category Mon^{Σ} of representations of Σ we consider the *associativity equation*, i.e., the pair of half-equations given by

$$\Theta^3 \xrightarrow{\ \Theta \times m\ } \Theta^2 \xrightarrow{\ m\ } \Theta$$
$$\Theta^3 \xrightarrow{\quad m \times \Theta \quad} \Theta^2 \xrightarrow{\ m\ } \Theta$$

Analogously, we define the left and right *identity equations* given by the morphisms

$$\Theta \xrightarrow{\ e \times \Theta\ } \Theta^2 \xrightarrow{\ m\ } \Theta \quad \text{and} \quad \Theta \xrightarrow{\ \Theta \times e\ } \Theta^2 \xrightarrow{\ m\ } \Theta$$
$$\Theta \xrightarrow{\qquad = \qquad} \Theta \qquad\qquad\qquad \Theta \xrightarrow{\qquad = \qquad} \Theta$$

and we denote by E the system constituted by these three equations.

The category $\mathrm{Mon}^{(\Sigma, E)}$ is the category of monads with a structure of monoid and (M, ρ) is its initial object.

6.2. **Lambda-calculus modulo α-equivalence.** We denote by $\Lambda(X)$ the set of lambda-terms up to α-equivalence with free variables "indexed" by the set X. It is well-known [BP99b, AR99, HM07] that Λ has a natural structure of monad where the monad composition is given by variable substitution.

It can be easily verified ([HM07]) that application and abstraction are Λ-linear natural transformations

$$\mathsf{app} := \Lambda^2 \longrightarrow \Lambda, \qquad \mathsf{abs} := \Lambda' \longrightarrow \Lambda.$$

that is, Λ is a monad endowed with a representation ρ of the signature $\Sigma = \{\mathsf{app} \colon \Theta^2 \to \Theta, \mathsf{abs} \colon \Theta' \to \Theta\}$.

Again, the monad Λ is initial in the category Mon^Σ of monads endowed representations of the signature Σ.

For our purposes, it is important to introduce another interesting, non-raw, representation on Λ which is defined as follows. Consider the Λ-module morphism

$$\mathsf{app}_1 \colon \Lambda \longrightarrow \Lambda'$$

given by $\mathsf{app}_1(x) = \mathsf{app}(x, *)$. Then app_1 is a representation of arity $\Theta \to \Theta'$ and the usual app constructor can be recovered from app_1 by *flattening*, i.e.,

$$\mathsf{app}(x, y) = \mathsf{eval}(\mathsf{app}_1(x), y).$$

Then we can consider on Λ the representation $\rho' = \{\mathsf{app}_1, \mathsf{abs}\}$ of signature $\Sigma' = \{\mathsf{app}_1 \colon \Theta \to \Theta', \mathsf{abs} \colon \Theta' \to \Theta\}$. The categories $\mathrm{Mon}^{\Sigma'}$ and Mon^Σ are equivalent through the *flattening* and (Λ, ρ') is the initial object of the category $\mathrm{Mon}^{\Sigma'}$.

6.3. Lambda-calculus modulo $\alpha\beta\eta$-equivalence.

Now we can introduce the lambda-calculus modulo $\alpha\beta\eta$-equivalence as a quotient of the previous calculus. We denote by $\mathsf{LC}(X)$ the set of lambda-terms up to $\alpha\beta\eta$-equivalence with free variables "indexed" by the set X. As in the previous example, the monad LC is endowed with a representation ρ and ρ' of the two signatures $\Sigma = \{\mathsf{app} \colon \Theta^2 \to \Theta, \mathsf{abs} \colon \Theta' \to \Theta\}$ and $\Sigma' = \{\mathsf{app}_1 \colon \Theta \to \Theta', \mathsf{abs}_1 \Theta' \to \Theta\}$ respectively. Now we want to introduce the equations for the β and η equivalence relations. This can be done both with ρ and ρ', but it looks more natural when it is formulated with respect to ρ'.

We define the β and η equivalence relation as the pair of half-equations on $\mathrm{Mon}^{\Sigma'}$ given by

$$\begin{array}{ccc} \Theta' \xrightarrow{\mathsf{abs}} \Theta \xrightarrow{\mathsf{app}_1} \Theta' & \quad\text{and}\quad & \Theta \xrightarrow{\mathsf{app}_1} \Theta' \xrightarrow{\mathsf{abs}} \Theta \\ \Theta' \xrightarrow{\quad = \quad} \Theta' & & \Theta \xrightarrow{\quad = \quad} \Theta \end{array}$$

respectively. We denote by E the system constituted by the β and η equations. It can be shown that (LC, ρ') is initial in the category $\mathrm{Mon}^{(\Sigma', E)}$ (this has been proved formally in the Coq proof assistant and discussed in [HM07]).

6.4. HOcore.

HOcore is a higher-order calculus for concurrency introduced by Lanese, Pérez, Sangiorgi and Schmitt [LPSS08]. The syntax of HOcore is as follows

$$P \quad ::= \quad x \quad | \quad a(x).P \quad | \quad \bar{a}.\langle P \rangle \quad | \quad P \parallel P \quad | \quad \mathbf{0}$$

where a ranges over the sort of *names* (or *channels*) and x denote a process variable. In the input prefix process $a(x).P$ the variable x is bound in the body P. In our framework the HOcore syntax is represented by the signature

$$\begin{aligned} \Sigma_{\mathsf{HOcore}} = \{ \quad & \forall a.\ \mathsf{recv}_a \colon \Theta' \to \Theta, \quad \forall a.\ \mathsf{send}_a \colon \Theta \to \Theta, \\ & \mathsf{parallel} \colon \Theta^2 \to \Theta, \quad \mathsf{zero} \colon * \to \Theta \quad \} \end{aligned}$$

where recv and send are families of constructors parametrized over the names a. Let us denote by HOcore the monad $\hat{\Sigma}_{\mathsf{HOcore}}$.

To spell out a concrete example, consider a pair of concurrent processes where the first process sends a message P through the channel a to the second process:

$$\bar{a}\langle P\rangle \mathbin{||} a(x).Q$$

In our formalism such process is represented by the syntax tree

$$\mathsf{parallel}(\mathsf{send}_a P, \mathsf{recv}_a Q)$$

where $P \in \mathsf{HOcore}(X)$ and $Q \in \mathsf{HOcore}(X+*)$ for some set of name variables X. Now consider the reduction rule

$$\bar{a}\langle P\rangle \mathbin{||} a(x).Q \quad \equiv \quad \mathbf{0} \mathbin{||} Q[x := P]$$

which models the exchange of a message P through a channel a between two concurrent processes. This can be stated as the equality of the two linear morphisms of kind $\mathsf{HOcore} \times \mathsf{HOcore}' \longrightarrow \mathsf{HOcore}$ given by

$$(P, Q) \mapsto \mathsf{parallel}(\mathsf{send}_a P, \mathsf{recv}_a Q)$$
$$(P, Q) \mapsto \mathsf{parallel}(\mathsf{zero}, \mathsf{eval}(Q, P))$$

Once abstracted over the representation ρ, the previous pair of morphisms gives a pair of half-equations on the category $\mathrm{Mon}^{\Sigma_{\mathsf{HOcore}}}$.

6.5. **Lambda calculus with explicit substitution.** We now consider an example of non algebraic signature. On the monad $\Lambda(X)$ of lambda calculus (modulo α-equivalence) given in 6.2 the substitution operator

$$\mathsf{subst} \colon \Lambda \cdot \Lambda \longrightarrow \Lambda$$

given by the monad composition (or *join*) is a representation of the arity $\Theta \cdot \Theta \to \Theta$ ($\Theta \cdot \Theta$ has a natural structure of module thanks to 2.2 (2)). We then have a representation ρ of the signature

$$\Sigma := \{ \quad \mathsf{app} \colon \Theta^2 \to \Theta, \quad \mathsf{abs} \colon \Theta' \to \Theta, \quad \mathsf{subst} \colon \Theta \cdot \Theta \to \Theta \quad \}.$$

In the monad Λ, substitution interacts with the app and abs constructors in the following way. Let us denote by $\mathsf{subst}' \colon \Theta \cdot \Theta' = (\Theta \cdot \Theta)' \to \Theta'$ the linear transformation induced by subst. Given any monad P we have a natural transformation $\mathsf{swap}_X \colon P(X) + * \longrightarrow P(X + *)$. Then (Λ, ρ) satisfies the system E constituted by the two equations

$$\Theta \cdot \Theta \times \Theta \cdot \Theta \xrightarrow{\;\mathsf{app}\cdot\Theta\;} \Theta \cdot \Theta \xrightarrow{\;\mathsf{subst}\;} \Theta$$
$$\Theta \cdot \Theta \times \Theta \cdot \Theta \xrightarrow[\mathsf{subst}\times\mathsf{subst}]{} \Theta \times \Theta \xrightarrow[\mathsf{app}]{} \Theta$$

and

$$\Theta' \cdot \Theta \xrightarrow{\;\mathsf{abs}\cdot\Theta\;} \Theta \cdot \Theta \xrightarrow{\;\mathsf{subst}\;} \Theta$$
$$\Theta' \cdot \Theta \xrightarrow[\Theta\cdot\mathsf{swap}]{} \Theta \cdot \Theta' \xrightarrow[\mathsf{subst}']{} \Theta' \xrightarrow[\mathsf{abs}]{} \Theta$$

We call Mon^{Σ} the category of lambda-calculi with *explicit substitution*. We claim that (Λ, ρ) is its initial object in $\mathrm{Mon}^{(\Sigma, E)}$.

6.6. **Lambda-calculus with explicit differentiation.** Inspired by [ER03] we sketch the definition of what we call *Lambda-calculus with explicit differentiation* (LCED). The syntax of LCED is given by the signature

$$\Sigma_{\mathsf{LCED}} := \{\quad \mathsf{app} \colon \Theta^2 \to \Theta, \quad \mathsf{abs} \colon \Theta' \to \Theta, \quad \mathbf{0} \colon * \to \Theta, \quad + \colon \Theta^2 \to \Theta,$$
$$\mathsf{partial} \colon \Theta' \times \Theta \to \Theta', \quad \forall k \in \mathbb{N}^*. \mathsf{diff}_k \colon \Theta^2 \to \Theta \quad \}.$$

We use the notations $\partial t \cdot u$ for $\mathsf{partial}(u, t)$ (the "linear" substitution — in the sense of [ER03] — of u for the first variable in t) and $D_k t \cdot u$ for $\mathsf{diff}_k(t, u)$ (the differential of t with respect to the k-th variable applied to u).

The link between differentiation and linear substitution is essentially given by the rule

$$D_1(\mathsf{abs}\, t) \cdot u = \mathsf{abs}\,(\partial t \cdot u)$$

The explicit linear substitution has its set of associated rules. E.g., for the substitution of an application we have

$$\partial(\mathsf{app}(s, t)) \cdot u = \mathsf{app}(\partial s \cdot u, t) + \mathsf{app}(D_1 s \cdot (\partial t \cdot u), t).$$

We omit the complete set of rules of the calculus which will be treated in a future work.

7. Related Works

The idea that the notion of monad is suited for modelling substitution concerning syntax (and semantics) has been retained by many recent contributions concerned with syntax (see e.g. [BP99a, GU03, MU04]) although some other settings have been considered. Notably in [FPT99] the authors work within a setting roughly based on operads (although they do not write this word down). Our main specificity here is the systematic use of the observation that the natural transformations we deal with are linear with respect to natural structures of module (a form of linearity had already been observed, in the operadic setting, see [FT01], section 4).

The signatures we consider here are slightly more general than the signatures in [FPT99], which allows us to give a signature to our app_1. On the other hand, our signatures 'reduce' (by *flattening*) to those in [FPT99]. In some future work, we plan to recover signatures as in [MU04] in terms of modules and to extend our initial semantics to this setting.

References

[ABF+05] B. Aydemir, A. Bohannon, M. Fairbairn, J. Foster, B. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic, *Mechanized metatheory for the masses: The POPLmark challenge*, Proceedings of the Eighteenth International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2005), 2005.

[AR99] Thorsten Altenkirch and Bernhard Reus, *Monadic presentations of lambda terms using generalized inductive types*, CSL, 1999, pp. 453–468.

[BP99a] Richard Bird and Ross Paterson, *Generalised folds for nested datatypes*, Formal Aspects of Computing **11** (1999), no. 2, 200–222.

[BP99b] Richard S. Bird and Ross Paterson, *De Bruijn notation as a nested datatype*, Journal of Functional Programming **9** (1999), no. 1, 77–91.

[ER03] Thomas Ehrhard and Laurent Regnier, *The differential lambda-calculus*, Theor. Comput. Sci. **309** (2003), no. 1, 1–41.

[FPT99]   Marcelo Fiore, Gordon Plotkin, and Daniele Turi, *Abstract syntax and variable binding*, LICS '99: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (Washington, DC, USA), IEEE Computer Society, 1999, p. 193.

[FT01]   Marcelo P. Fiore and Daniele Turi, *Semantics of name and value passing*, Logic in Computer Science, 2001, pp. 93–104.

[GU03]   Neil Ghani and Tarmo Uustalu, *Explicit substitutions and higher-order syntax*, MERLIN '03: Proceedings of the 2003 ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding (New York, NY, USA), ACM Press, 2003, pp. 1–7.

[HM07]   André Hirschowitz and Marco Maggesi, *Modules over monads and linearity*, WoLLIC (Daniel Leivant and Ruy J. G. B. de Queiroz, eds.), Lecture Notes in Computer Science, vol. 4576, Springer, 2007, pp. 218–237.

[LPSS08]   Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt, *On the expressiveness and decidability of higher-order process calculi*, Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008), June 2008, To appear.

[MU04]   Ralph Matthes and Tarmo Uustalu, *Substitution in non-wellfounded syntax with variable binding*, Theor. Comput. Sci. **327** (2004), no. 1-2, 155–174.

[Sel96]   P. Selinger, *The lambda calculus is algebraic*, 1996.

[Zsi06]   Julianna Zsidó, *Le lambda calcul vu comme monade initiale*, Master's thesis, Université de Nice – Laboratoire J. A. Dieudonné, 2005/06, Mémoire de Recherche – master 2.